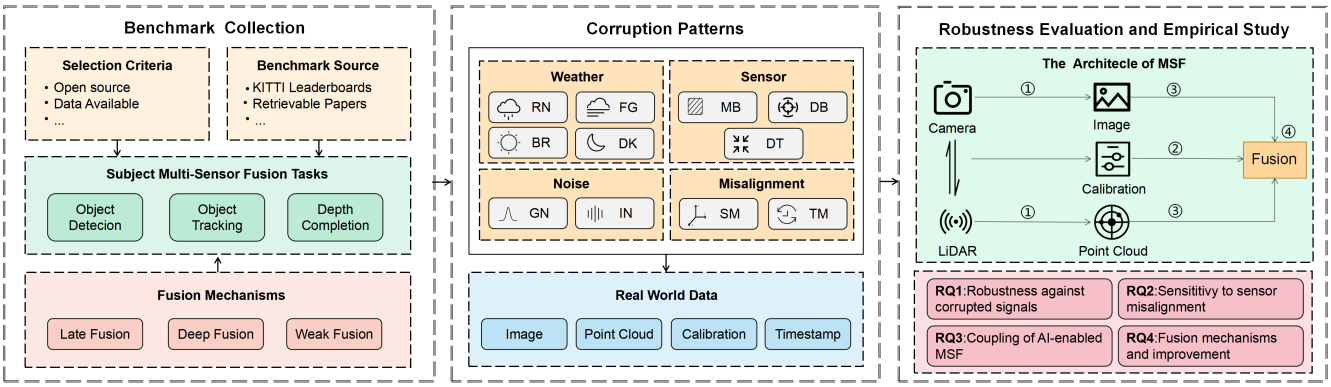
AI-MSF-Benchmark

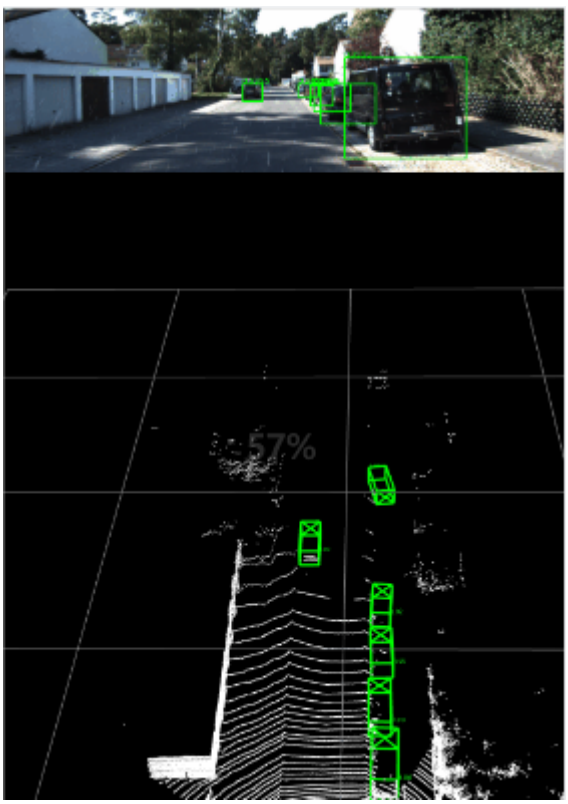
This repository provides the code of the paper "**An Empirical Study of AI-enabled Multi-Sensor Fusion Based Perception Systems**".

[website]



AI-enabled MSF systems. we summarizes the seven MSF systems used in our benchmark in this link. These seven systems cover three different tasks and three different fusion mechanisms

Corruption. We leverage fourteen common corruption patterns to synthesize corrupted data that could possibly occur in the operational environments to evaluate MSF systems' robustness. these corruption patterns can be naturally grouped into four categories: weather corruption, sensor corruption, noise corruption, and sensor misalignment. we visualize the corruption patterns in this link.



Installation

We implement all the MSF systems with PyTorch 1.8.0 and Python 3.7.11. All experiments are conducted on a server with an Intel i7-10700K CPU (3.80 GHz), 48 GB RAM, and an NVIDIA GeForce RTX 3070 GPU (8 GB VRAM).

Corruption Patterns Dependency

Run the following command to install the dependencies

```
cd Corruption
pip install -r requirements.txt
```

If you need to synthesize rain and fog, you need to install the following dependencies For synthesize realistic images of rain and fog, you need to install the dependencies in this repository. link

- optional : git clone the repository and move to AI-MSF-Benchmark/corruption/3rd_parts/camera/
- Install dependencies
- generate depth maps
- download particles simulation files
- construct datasets to the required format.

```
corruption/3rd_parts/camera/data/source/kitti/data_object/training/image_2/file0001.png
      # Source images (color, 8 bits)
corruption/3rd_parts/camera/data/source/kitti/data_object/training/image_2/depth/file0001.p
ng     # Depth images (16 bits, with depth_in_meter = depth/256.)
corruption/3rd_parts/camera/data/source/particles/kitti/data_object/rain/10mm/*.xml
      # Particles simulation files (here, 10mm/hr rain)
```

For synthesize realistic point cloud of rain and fog, you need to install the dependencies in this repository. link

- optional : git clone the repository and move to AI-MSF-Benchmark/corruption/3rd_parts/lidar/
- Install dependencies

Install MSF-based Systems

In order to reproduce our experiments, we need to carefully configure the environment for each system. The details are here

The structure of the repository

Folder Structure:

```
AI-MSF-Benchmark
|-system            seven AI-enabled MSF-based perception systems
|-corruption        fourteen corruption patterns
|-utils
|-tools
```

Usage

Synthesize Corrupted Dataset

Synthesize corrupted data for KITTI object detection dataset.

Applying corruption operators to a clean dataset (e.g. KITTI) can synthesize realistic corrupted dataset. Note that corruption patterns used in this study can also generalize to other datasets, such as Waymo, NuScenes. The corruption patterns we offer can be used separately.

Run the following command to synthesize corruption image:

```
python -m corruption.main simulate_image --input_dir "input path" --output_dir "output
path" --corruption {corruption} --sev {severity level}
```

Similarly, we also provide commands for synthesizing other types of corruption data

```
python -m simulate_image/simulate_lidar/simulate_calib/simulate_weather/simulate_delay
params1 ... prams2 ...
```

Here are some examples:

Add brightness with a severity level of 3 to the image.

```
python -m corruption.main simulate_image \
--input_dir "./corruption/example/input_dir/image" \
--output_dir "./corruption/example/output_dir/image/" \
--corruption "brightness" --sev 3
```

Add Gaussian noise with a severity level of 5 to the point cloud.

```
python -m corruption.main simulate_lidar \
--input_dir "./corruption/example/input_dir/velodyne" \
--output_dir "./corruption/example/output_dir/velodyne/" \
--corruption "gaussian_noise" --sev 5
```

Add 2 degree rotation to x rotation axis to simulate spatial misalignment between the camera and LiDAR.

```
python -m corruption.main simulate_calib \
--input_dir "./corruption/example/input_dir/calib" \
--output_dir "./corruption/example/output_dir/calib/" \
--corruption "rotation_x" --sev 3
```

Add 0.1 second (i.e. 1frame) delay on camera.

```
python -m corruption.main simulate_delay \
--input_dir "./corruption/example/input_dir/sequence" \
--output_dir "./corruption/example/output_dir/delay_image/" \
--corruption "delay" --sev 1
```

Add rain with 10 mm/h rainfall to images and point clouds. (Note that you need to correctly configure third-party dependencies in "Corruption Patterns Dependency" section)

```
python -m corruption.main simulate_weather \
--input_dir_c "./corruption/example/input_dir/image" \
--output_dir_c "./corruption/example/output_dir/rain/image/" \
--input_dir_l "./corruption/example/input_dir/velodyne" \
--output_dir_l "./corruption/example/output_dir/rain/velodyne/" \
--calibdir "./corruption/example/input_dir/calib" \
--depthdir "./corruption/example/input_dir/depth" \
--corruption "rain" --sev 1
```

Refer to `corruption/corruption_configs.py` to learn more details about the corruptions.

Evaluate MSF-based Systems

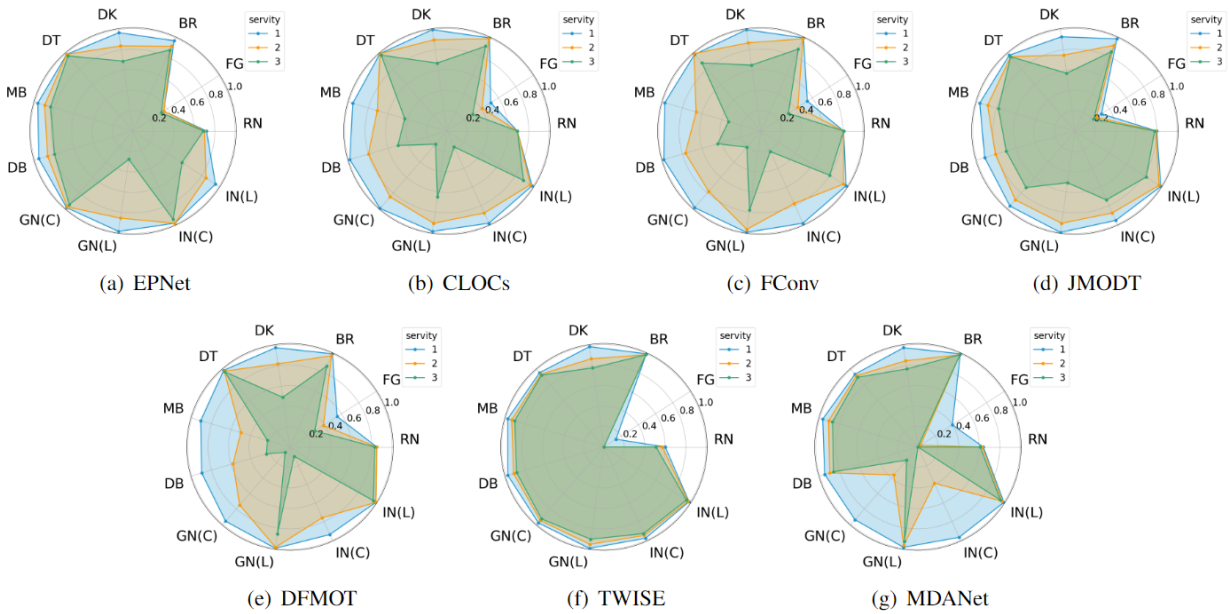
In order to reproduce our experiments(RQ1, RQ2, RQ3), you should

1. Install the dependencies according to the requirements of each system (in Sec. *Install MSF-based Systems*).
2. Place the specified synthetic data set in the corresponding directory for each system.
3. Evaluate the system performance.

In order to reproduce RQ4, for CLOCs-Rb, run `tools/aggerate_bounding_box.py` to aggregate high confidence and unique results from an individual branch to the fusion results. For FConv, you should Generate the 2D front view images of rgb images.

1. Train a 2D detector on 2D front view images.
2. Copy Second from `AI-MSF-benchmark/system/CLOCs/second` to `AI-MSF-benchmark/ops/second`.
3. Run `tools/aggerate_bounding_box.py` to aggregate guidance from different branch.

Results



(a) EPNet (b) CLOCs (c) FConv (d) JMODT

(e) DFMOT (f) TWISE (g) MDANet